JASPERREPORTS - INTERNATIONALIZATION

At times we need reports in different languages. Writing the same report for each different language implies a lot of redundant work. Only pieces of text differing from language to language should be written separately, and loaded into text elements at runtime, depending on locale settings. This is the purpose of the report internationalization. Internationalized reports once written can run everywhere.

In the following steps, we have listed how to generate a report in different languages and also some other features of report internationalization:

- Associate a resource bundle *java.util.ResourceBundle* with the report template. There are two ways to associate the java.util.ResourceBundle object with the report template.
    - At design time, by setting the *resourceBundle* attribute of the report template object to the base name of the target resource bundle.
    - A dynamic/runtime association can be made by supplying a *java.util.ResourceBundle* object as the value for the REPORT_RESOURCE_BUNDLE parameter at report-filling time.
    - If the report needs to be generated in a locale that is different from the current one, the built-in REPORT_LOCALE parameter can be used to specify the runtime locale when filling the report.
- To facilitate report internationalization, a special syntax **$R{}** is available inside report expressions to reference *java.lang.String* resources placed inside a *java.util.ResourceBundle* object associated with the report. The **$R{}** character syntax extracts the locale-specific resource from the resource bundle based on the key that must be put between the brackets:

    ```
    <textFieldExpression>
    $R{report.title}
    </textFieldExpression>
    ```

    The above text field displays the title of the report by extracting the String value from the resource bundle associated with the report template based on the runtimesupplied locale and the *report.title* key
- Formatting messages in different languages based on the report locale, there's a built-in method inside the report's *net.sf.jasperreports.engine.fill.JRCalculator*. This method offers functionality similar to the *java.text.MessageFormat* class. This method, msg(), has three convenient signatures that allow you to use up to three message parameters in the messages.
- A built-in *str()* method (the equivalent of the $R{} syntax inside the report expressions), which gives access to the resource bundle content based on the report locale.
- For date and time formatting, the built-in REPORT_TIME_ZONE parameter can be used to ensure proper time transformations.
- In the generated output, the library keeps information about the text run direction so that documents generated in languages that have right-to-left writing (like Arabic and Hebrew) can be rendered properly.
- If an application relies on the built-in Swing viewer to display generated reports, then it needs to be internationalized by adapting the button ToolTips or other texts displayed. This is very easy to do since the viewer relies on a predefined resource bundle to extract locale-specific information. The base name for this resource bundle is *net.sf.jasperreports.view.viewer*

Example

To demonstrate internationalization, let's write new report template (jasper_report_template.jrxml). The contents of the JRXML are as below. Save it to C:\tools\jasperreports-5.0.1\test directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport xmlns="http://jasperreports.sourceforge.net/jasperreports"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperreports
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="jasper_report_template" language="groovy" pageWidth="595"
pageHeight="842" columnWidth="555" leftMargin="20" rightMargin="20"
topMargin="20" bottomMargin="20" resourceBundle="localizationdemo">
   <title>
      <band height="552">
         <textField>
            <reportElement positionType="Float"
            x="150" y="20"
            width="400" height="50"/>
            <textElement>
               <font size="24"/>
            </textElement>
            <textFieldExpression
            >
            <![CDATA[$P{REPORT_LOCALE}.getDisplayName
            ($P{REPORT_LOCALE})]]>
            </textFieldExpression>
         </textField>

         <textField isStretchWithOverflow="true"
         isBlankWhenNull="true">
            <reportElement positionType="Float" x="20" y="125"
            width="530" height="20"/>
            <textElement textAlignment="Justified">
               <font size="14"/>
            </textElement>
            <textFieldExpression >
            <![CDATA[$R{localization.text1}]]>
            </textFieldExpression>
         </textField>
      </band>
   </title>
</jasperReport>
```

In the above file, the *resourceBundle* attribute of the <jasperReport> element tells JasperReports where to get the localized strings to use for the report. We need to write a property file with a root name matching the value of the attribute must exist anywhere in the CLASSPATH when filling the report. In this example the property file **localizationdemo.properties** is saved under the directory **C:\tools\jasperreports-5.0.1\test**. The contents of this file are as below

```
localization.text1=This is English text.
```

To use a different locale, the name of the file must be localizationdemo_[locale].properties. Here we will write a file for spanish locale. Save this file as : **C:\tools\jasperreports-5.0.1\test\localizationdemo_es.properties.**. The contents of this file are as below:

```
localization.text1=Este texto es en Español.
```

The syntax to obtain the value for resourceBundle properties is $R{key}.

To let JasperReports know what locale we wish to use, we need to assign a value to a built-in parameter. This parameter's name is defined as a constant called REPORT_LOCALE, and this constant is defined in the

net.sf.jasperreports.engine.JRParameter class. The constant's value must be an instance of *java.util.Locale*. This logic is incorporated in java code to fill and generate the report. Let's save this file **JasperReportFillI18.java** to C:\tools\jasperreports-5.0.1\test\src\com\tutorialspoint directory. The contents of the file are as below:

```
package com.tutorialspoint;

import java.util.HashMap;
import java.util.Locale;

import net.sf.jasperreports.engine.JREmptyDataSource;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JRParameter;
import net.sf.jasperreports.engine.JasperFillManager;

public class JasperReportFillI18 {

   public static void main(String[] args) {
      String sourceFileName = "C://tools/jasperreports-5.0.1/test/"
            + "jasper_report_template.jasper";
      HashMap parameterMap = new HashMap();
      if (args.length > 0) {
         parameterMap.put(JRParameter.REPORT_LOCALE, new Locale(args[0]));
      }
      try {
         JasperFillManager.fillReportToFile(sourceFileName, null,
               new JREmptyDataSource());
      } catch (JRException e) {
         // TODO Auto-generated catch block
         e.printStackTrace();
      }

   }
}
```

Report generation

We will compile and execute the above file using our regular ANT build process. The contents of the file build.xml (saved under directory C:\tools\jasperreports-5.0.1\test) are as below.

> *The import file - baseBuild.xml is picked from chapter Environment Setup and should be placed in the same directory as the build.xml.*

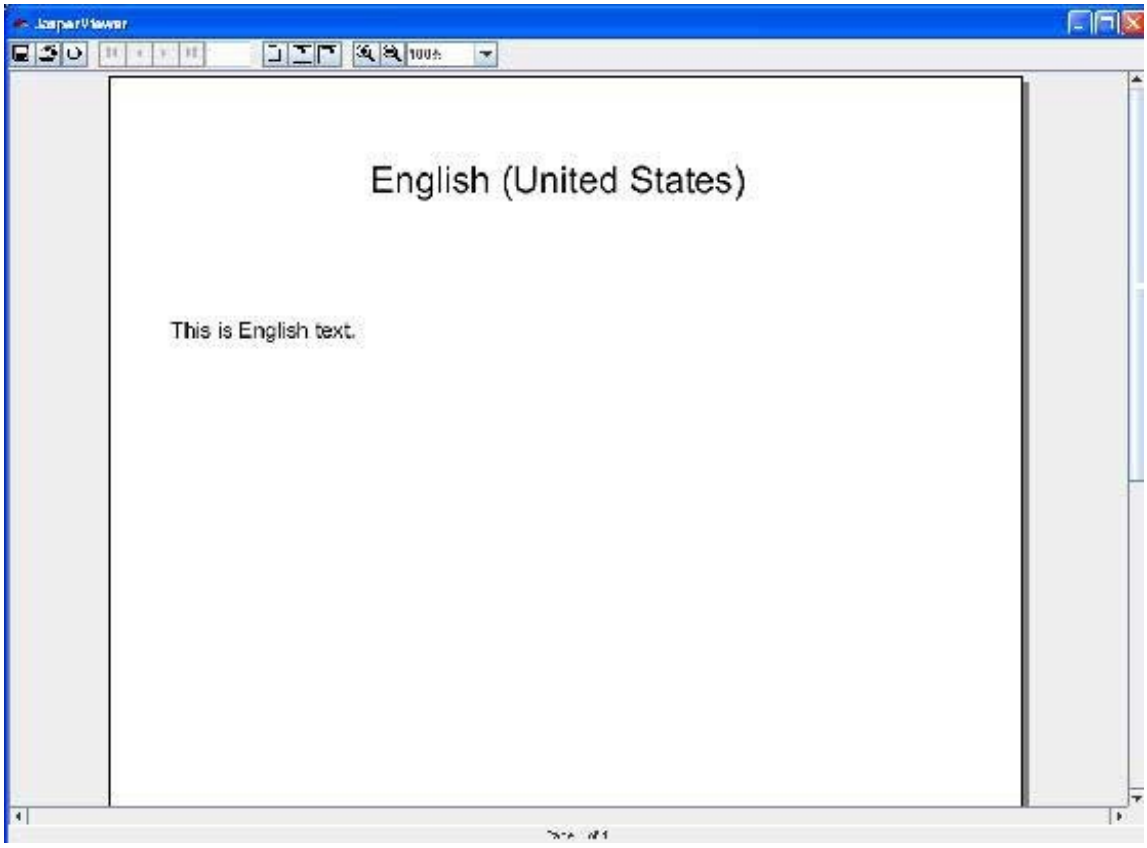
```
<?xml version="1.0" encoding="UTF-8"?>
<project name="JasperReportTest" default="viewFillReport" basedir=".">
   <import file="baseBuild.xml" />
   <target name="viewFillReport"
      depends="compile,compilereportdesing,run"
      description="Launches the report viewer to preview
      the report stored in the .JRprint file.">
      <java classname="net.sf.jasperreports.view.JasperViewer"
      fork="true">
         <arg value="-F${file.name}.JRprint" />
         <classpath ref />
      </java>
   </target>
   <target name="compilereportdesing"
      description="Compiles the JXML file and
      produces the .jasper file.">
      <taskdef name="jrc"
      classname="net.sf.jasperreports.ant.JRAntCompileTask">
         <classpath ref />
      </taskdef>
      <jrc destdir=".">
         <src>
```

```
         <fileset dir=".">
            <include name="*.jrxml" />
         </fileset>
         </src>
         <classpath ref />
      </jrc>
   </target>
</project>
```

Next, let's open command line window and go to the directory where build.xml is placed. Finally execute the command **ant -Dmain-class=com.tutorialspoint.JasperReportFillI18** (viewFullReport is the default target) as follows:

```
C:\tools\jasperreports-5.0.1\test>ant  -Dmain-class=com.tutorialspoint.JasperReportFillI18
Buildfile: C:\tools\jasperreports-5.0.1\test\build.xml

clean-sample:
   [delete] Deleting directory C:\tools\jasperreports-5.0.1\test\classes
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jasper
   [delete] Deleting: C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrprint

compile:
    [mkdir] Created dir: C:\tools\jasperreports-5.0.1\test\classes
    [javac] C:\tools\jasperreports-5.0.1\test\baseBuild.xml:28:
    warning: 'includeantruntime' was not set, defaulting to
    [javac] Compiling 1 source file to C:\tools\jasperreports-5.0.1\test\classes
    [javac] Note: C:\tools\jasperreports-
5.0.1\test\src\com\tutorialspoint\JasperReportFillI18.java
    uses unchecked or u
    [javac] Note: Recompile with -Xlint:unchecked for details.

compilereportdesing:
      [jrc] Compiling 1 report design files.
      [jrc] log4j:WARN No appenders could be found for logger
      (net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
      [jrc] log4j:WARN Please initialize the log4j system properly.
      [jrc] log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
      [jrc] File : C:\tools\jasperreports-5.0.1\test\jasper_report_template.jrxml ... OK.

run:
     [echo] Runnin class : com.tutorialspoint.JasperReportFillI18
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

viewFillReport:
     [java] log4j:WARN No appenders could be found for logger
     (net.sf.jasperreports.extensions.ExtensionsEnvironment).
     [java] log4j:WARN Please initialize the log4j system properly.

BUILD SUCCESSFUL
Total time: 3 minutes 28 seconds
```

As a result of above compilation, a JasperViewer window opens up as in the screen below:


English (United States)
This is English text.